

Abgabe Übungsblatt 02

Technische Informatik 2 Wintersemester 2025/26

Baris Basar, Cengizhan Evcil, Nicolai Held

Aufgabe 1

Umleitung

Gib die korrekten Ein-/Ausgabeumleitungen an.

- a) Es wird das Kommando **lto-dump -14** ausgeführt. Umlenken der Standardausgabe in einen Prozess, der das Kommando **logilab-pytest** ausführt.
- b) Es wird das Kommando **llvm-nm -14** ausgeführt. Umlenken der Datei **hypothesis-corn** in die Standardeingabe des Prozesses.
- c) Es wird das Kommando **ps2pk** ausgeführt. Die Standardausgabe wird in die Datei **notebook** umgelenkt. Existiert die Datei bereits, wird sie geleert.
- d) Es wird das Kommando **cmake** ausgeführt. Die Fehlerausgabe wird in die Datei **premium-transfer** umgelenkt. Existiert die Datei bereits, wird sie geleert.
- e) Es wird das Kommando **pbtntext** ausgeführt. Anhängen der Ausgaben auf dem Standardausgabekanal an die Datei **widen_brown**. Umlenken der Standardausgabe in einen Prozess, der das Kommando **elfedit -c** ausführt.
- f) Es wird das Kommando **db5_3_tuner** ausgeführt. Anhängen der Ausgaben auf dem Fehlerkanal an die Datei **scale-freckle**. Umlenken der Datei **premium-transfer** in die Standardeingabe des Prozesses.

Absenden

Flag: flag{b3bd6c6d-6064-4ac3-a6b3-aec8d075f5e40}

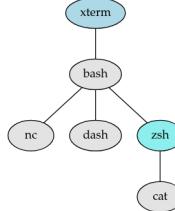
- a) Ausgabe soll direkt Eingabe des nächsten Kommandos sein → Pipe
- b) STDIN umleiten (<)
- c) STDOUT überschreiben (>)
- d) STDERR Fehlerkanal = 2>
- e) Zuerst anhängen an Datei (»), danach Pipe für Prozess
- f) STDERR anhängen (2») + STDIN umleiten (<)

Aufgabe 2

```

bash
nc -l 5679 &
ls -l &
fg &
ctrl-z
dash &
zsh
ls -la &
jobs
cd -
yes > /dev/null &
kill %1
ping4 127.0.0.1 &
fg
ctrl-c
cat /dev/zero &

```



Flag: flag{60eaae80-75ba-4540-8908-a25146ad1e3c}

Füllt für jeden Prozess, der nach dem letzten Kommando noch ausgeführt wird, eine Zeile aus. Geht jeweils den Prozess und seinen Elternprozess an (jeweils ohne ihre Parameter). Gebt außerdem an, ob der Prozess im Hintergrund ausgeführt wird (j) oder nicht (n).

Prozessname:	bash	Elternprozess:	xterm	Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:		Elternprozess:		Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:	nc	Elternprozess:	bash	Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:	dash	Elternprozess:	bash	Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:	zsh	Elternprozess:	bash	Hintergrundprozess (j/n)?:	<input checked="" type="checkbox"/>
Prozessname:		Elternprozess:		Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:		Elternprozess:		Hintergrundprozess (j/n)?:	<input type="checkbox"/>
Prozessname:		Elternprozess:		Hintergrundprozess (j/n)?:	<input type="checkbox"/>

Um die Aufgabe vernünftig zu lösen ist es das einfachste, wenn man die entsprechenden Befehle direkt selbst im Terminal eingibt. Nachdem man dies erfolgreich getan hat, kann man mit dem Befehl "pstree" den Prozessbaum einsehen:

```

      125*[{firefox}]
      gjs—11*[{gjs}]
      vpnui—7*[{vpnui}]
      42*[{gnome-shell}]
      gnome-shell-cal—5*[{gnome-shell-cal}]
      gnome-terminal—gnome-terminal.—3*[{gnome-term
      gnome-terminal.—bash—bash—dash
          nc
          zsh—cat
          pstree
      4*[{gnome-terminal-}]
      goa-daemon—3*[{goa-daemon}]
      goa-identity-se—2*[{goa-identity-se}]
      gsd-a11y-settin—3*[{gsd-a11y-settin}]
      gsd-color—3*[{gsd-color}]
```

Um zu wissen, welche Prozesse im Vordergrund ausgeführt werden und welche im Hintergrund ausgeführt werden, muss man sich den Befehlen *fg* und *bg* (Vordergrund und Hintergrund) bewusst sein, außerdem ist zu beachten, dass, wenn ein Befehl mit "&" am Ende gestartet wird, dieser Befehl im Hintergrund ausgeführt wird. Wenn in einer Shell wie "bash" eine neue Shell "zsh" ausgeführt wird, dann wird die alte Shell zum Hintergrundprozess und die neue zum

Vordergrundprozess.

Aufgabe 3

Signalserver für SIGCONT

Beschreibung

Dieses Programm wartet auf Signale des Typs **SIGCONT** und gibt Informationen über den Sender aus.

Es verwendet **sigaction** mit **SA_SIGINFO**, um die PID und UID des sendenden Prozesses zu ermitteln.

Aufbau

- `signalHandler(int signum, siginfo_t* info, void* ctx):`
 - Gibt die Signalnummer aus.
 - Gibt PID und UID des Senders aus (wenn verfügbar).
 - Meldet, dass das Programm weiterläuft.
- `main():`
 - Initialisiert **sigaction**.
 - Setzt Handler für **SIGCONT**.
 - Wartet in einer Endlosschleife auf Signale (**pause()**).

Nutzung

1. Kompilieren:

```
g++ -o sigserver sigserver.cpp
```

2. Starten:

```
./sigserver
```

3. Signal senden (von anderem Terminal):

```
kill -SIGCONT <PID_des_Programms>
```

Beispieldaten: sigserver gestartet. PID = 12345 Warten auf SIGCONT... Signal erhalten: 18 Sender-PID: 67890 Sender-UID: 1000 Programm läuft weiter...

Literaturverzeichnis

- 1) OpenAI ChatGPT Chatverlauf (s. Anhang) (04.11.2025)
- 2) Woyciechowski, Marcus (2020), Linux Cheat Sheet / Spickzettel: Übersicht der wichtigsten Terminal Befehle, in: woytec.de, 13.06.2020, <https://woytec.de/linux/linux-cheat-sheet-spickzettel-uebersicht-der-wichtigsten-terminal-befehle/>

- 3) Kerrisk, Michael (2025), `sigaction(2)` — Linux manual page, in: man7.org, 17.05.2025, <https://man7.org/linux/man-pages/man2/sigaction.2.html>

Anhang

Chatverlauf mit [ChatGPT] (<https://chat.openai.com/chat>) am 04.11.2025:
Frage: Zeig mir alle Ein-/Ausgabeumlenkungen

Operator Beschreibung

- > Standardausgabe (stdout) **in Datei schreiben** – Datei wird **neu angelegt oder geleert**
 - >> Standardausgabe **an Datei anhängen**
 - < Datei als **Standardeingabe (stdin)** verwenden
 - << **Here-Dокумент**: mehrere Zeilen als Eingabe im Skript
 - <<< **Here-String**: Zeichenkette als Eingabe
-

Operator Beschreibung

- 2> Standardfehler (stderr) in Datei schreiben – Datei **wird überschrieben**
 - 2>> Standardfehler an Datei **anhängen**
 - &> stdout und stderr zusammen in eine Datei (überschreiben)
 - &>> stdout und stderr **anhängen**
 - 2>&1 stderr auf stdout umleiten (oder umgekehrt)
 - 1>&2 stdout auf stderr umleiten
-

Beispiel	Beschreibung
<code>cmd >out 2>err</code>	stdout in out, stderr in err
<code>cmd >file 2>&1</code>	stdout und stderr in dieselbe Datei
<code>cmd >>file 2>&1</code>	anhängen an dieselbe Datei

Operator	Beschreibung
<code>cmd1 cmd2</code>	
<code>cmd1 \ & cmd2</code>	stdout und stderr per Pipe an cmd2
<code>cmd <(cmd2)</code>	Process substitution – Eingabe wie Datei
<code>cmd >(cmd2)</code>	Ausgabe wie Datei weiterleiten

Beispiel	Beschreibung
cmd 2>/dev/null	Fehlerausgabe verwerfen
cmd 1>&-	stdout schließen
cmd 2>&-	stderr schließen